International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Distributed Pattern Matching: Cycle-Based Query Optimization

Rajshri G. Deshmukh[1], Praful B. Sambhare[2]

[1, 2] Department of Computer Science & Engineering, P. R. Pote (Patil) College of Engineering & Mgmt, Amravati, Maharashtra, India

Abstract: ***Due to rapid growth of the Internet technology and new scientific/technological advances, the number of applications that model data as graphs increases, because graphs have high expressive power to model complicated structures. Greedy algorithms for subgraph pattern matching operations are often sufficient when the graph data set can be held in memory on a single machine. However, as graph data sets increasingly expand and require external storage and partitioning across a cluster of machines, more sophisticated query optimization techniques become critical to avoid explosions in query latency. In this paper, there is query optimization technique for distributed graph pattern matching***

Keywords: Subgraph, Pattern matching, Query processing, Subgraph isomorphism, Distributed pattern matching, Graph simulation.

1. Introduction

The graph data model is becoming an increasingly popular way to represent data for various applications. Reasons for this include:

1) It can be less complex for a user to load semi-structured or sparse data into a vertex-edge-vertex data.
2) Some increasingly popular data sets or networks(such as the Twitter, Facebook social networks) are most preferable for using a graph structures.
3) Graph operations like shortest path calculations, subgraph pattern matching, and PageRank are easily expressed over a graph data model [3].

Many large graph data sets are difficult to manage on a single machine. Major network systems are unable to manage on to the single memory unit and therefore clusters of machines i.e. collections are being deployed for operations like process, store, manage, and analyze graph data. For instance, as of 2012, Facebook's user graph has 900 million vertices. In Semantic web community, the linking open data movement has collected 6 billion triples i.e. a triple is equivalent to an edge in a graph, from 300 interconnected data sets. There are various graph algorithms were designed with the consideration that the whole graph can be stored in memory on a single machine, these distributed architectures require revisiting these algorithms in a distributed context manner, and the factors like network latency and throughput can replace the traditional implementation of these algorithms. Graph pattern matching is fundamental to social network analysis. Traditional techniques are subgraph isomorphism and graph simulation [5]. Graph pattern-matching is a generalization of string matching, query matching and two-dimensional pattern-matching that others framework for the study of matching problems upon multi-dimensional architectures.

1.1 Graph Pattern Matching

Graph pattern matching is being mostly used in various applications, e.g., software plagiarism detection (means work or idea of someone else and pretend it is one's own), protein interaction networks, social networks and intelligence analysis. Pattern-matching on graphs is the problem of finding a homomorphic or isomorphic image of a given graph, called the pattern, in another graph, called the target. Since the image of the pattern is then a subgraph of the target, the problem is also known as the subgraph isomorphism problem. Graph matching is typically defined in terms of subgraph isomorphism [3].

Graph pattern matching, as an important class of graph queries, searches to find subgraphs of a data graph that are similar to a given query graph [14]. This problem has been extensively studied over the past several decades; however, the large scales of new application domains such as social networks and the World Wide Web have recreated interest in highly scalable graph pattern matching algorithms [15].

General Definition

The inputs of the subgraph pattern matching problem are two graphs. Two graphs are inputted to pattern matching algorithm. Given below:

1) A graph G with nodes V and edges E, where nodes and edges are labeled with characters. We denote the label of a node v and an edge e as label (v) and label (e), respectively.
2) A query pattern, which can be defined as a graph P = (Vp;Ep). Here, the nodes and edges describe some conditions so that a subgraph of G must satisfy in order to be a match. Continually, the query pattern is a conjunction of smaller patterns that together force requirements on nodes and their neighborhoods in the data graph G. Given a graph G, the pattern matching problem is to find all possible subgraphs of G that match a given pattern P[1].

Subgraph pattern matching is a most important operation that must be revisited for distributed graph stores. Subgraph matching operations are particularly used in social network data mining operations. The goal of graph pattern matching is to find all subgraphs of given graph, called the data graph.

And that are similar to a query graph in a structural and a semantic way [4].

Another approach for implementing graph pattern matching in large social network graphs is to employ distributed algorithms. One method of this is to partition a large graph G into fragments and then distribute the fragments across different sites, implement into networks [11]. Given a pattern graph Q, we partially compute Q over these fragments in parallel way, and then assemble the results to get the set M (Q, G) of matches for Q in the whole graph G. That is, we divide a large computational task into smaller ones of scalable sizes, and establish parallel processing to perform the computation. In fact number of large real-life graphs are already deployed and shared in different sites e.g., social networks, Web services networks [12]. It is natural way to perform distributed graph pattern matching by using partial computation. The partial computation techniques yield distributed query evaluation algorithms with several performance guarantees:

- Each site is visited a fixed number of times.
- The communication cost is determined by the fragmentation of graph G and the size of Q.
- The computational cost is determined by Q and the largest fragment of graph G in the partition [10].

1.2 Query Optimization

Query optimization in traditional database systems can be viewed as a search method that consists of three parts, search space generation, cost estimation and search. The query optimizer defines a space of query plans, explores the space with a search algorithm and estimates the cost of plans encountered. The problem of subgraph pattern matching can be same as, where the subgraph pattern being matched is the "query" that is processed over the raw graph data in a given graph [2].

There is defined one query optimization framework for subgraph pattern matching. In particular, given a data store represented in the graph data model. A query that request all instances of a particular graph pattern within the data store. When we apply algorithm that generate a series of query execution plans, compute the cost of each generated plan, and select the plan with lowest cost for execution. So here defined the cycle - based optimization [7]-[8].

Cycles appear more frequently in query patterns over a graph model than data represented in other models. Graph contains cycle structures than any other data model. Because in the network structure mostly appear cycles. They can be potentially and structurally beneficial to improve query performance of graph. It should be explicitly considered during query plan generation [6].The intention of using this algorithm is to remove cycles from graph and reduce the size of graph. So that it leads to attain high performance with less complexity. The cycle -based optimization framework based on the technique in which cycles tend to significantly reduce the size of intermediate result [13].

2. Cycle - Based Optimization

This section defines optimization framework which is based on cycle detection. However, if the graph contains cycles, it may be beneficial to match cycles first [9]. This is because matching a cycle serves as a type of selection predicate; it prevents the subset of the raw data set that needs to be explored. Therefore, matching cycle patterns first may lead to smaller intermediate results and consequently yield better better performance. For example, given a query A→B, A→C, B→A, it may be directed or undirected graph a cycle-based algorithm would match A→B, B→A first. While the algorithms that are presented in the previous section it first match A→B, A→C.

However, if a vertex in the raw data graph has 20 outgoing edges, then there are 190 matches to the A→B, A→C join. However, from 10 only 2 outgoing edges from that vertex have a return edge in the other direction, then joining A→ B, B→A first it reduces the intermediate result. The mechanism is it first search for a cycle then reduces it to form simple intermediate result set. One important note that the both directed and undirected cycles are able to reduce the size of intermediate result sets. The following provides the algorithm for cycle-based detection.

3. Algorithm

Figure1 shows the pseudo code for the cycle detection based optimization framework. The algorithm first converts the graph structure into an undirected graph and searches all cycles. After identification of cycle there are two approaches are employed to combine them.

The first approach is a greedy approach. In this it selects a cycle as a starting point and then keeps adding overlapping cycles to it greedily. These overlapping cycles are two cycles which share at least one edge (see Example 1). If overlapping cycles are not found, instead of that non-overlapping cycles are added. These overlapping cycles has high preference than non-overlapping cycles because it is more efficient to add overlapping cycles than non-overlapping cycles because part of the overlapping cycles has already been matched.

Then second approach is a bushy approach. The algorithm decomposes the graph structure into several fragments, finds matches in each fragment differently, and then combines the intermediate results of these fragments. A heuristic function is used for decomposition in which overlapping cycles remain in the same component. The reason for this heuristic is that overlapping cycles should produce smaller intermediate output than non-overlapping cycles. So it is best to group the overlapping cycles together in same component. Example 2 illustrates how the cycle detection-based optimization works.

function CycleDetectionQueryOptimization (G)
first convert directed query graph G into an undirected graph.
find all cycles using depth-first search on G
(perform DFS)
// greedy approach

for each cycle C found
current cycle = C
matched= ∅; // subgraph will be matched
while (current cycle |= null)
add current cycle to *matched* and compute the cost
if (there is a cycle *C0* overlapping with matched)
current cycle = *C0*
else if (there is a cycle *C1* left)
current cycle = *C1*
else
current cycle = null
// for edges that not present in cycles
if (there are any edges left)
add them to *matched* and compute the cost
// bushy approach
decompose G into fragments such that overlapping
cycles remain in the same fragment
for each decomposition D
find the cost of matching each fragment in D
find the costs of joining them
return the plan that having lowest cost in both approaches.

Figure1: Cycle – based algorithm

4. Example

Example 1. We use Q1, Q2 and Q3 in Figure 2 to illustrate the concept of (non-)overlapping cycles. In Q1, cycles ABC and DE do not overlap because they don't share any edges. In Q2, cycles AB and CBD do not overlap. They share vertex B, but do not share any edges. In Q3, cycles BD and BDE overlap because they share edge BD.

Example 2. There are three cycles in Q2 of Figure 2, namely, AB, BCD and BCE. Cycles BCD and BCE overlap by sharing edge BC. For the greedy approach, if it first matches BCD, it next adds edges B→E and C→E because BCE and BCE overlap. Finally, it adds edges A→B and B→A. For the bushy approach, the pattern is decomposed into two fragments. The first one contains cycle AB and the second has cycles BCD and BCE. Matching is done on two fragments separately and results are joined.

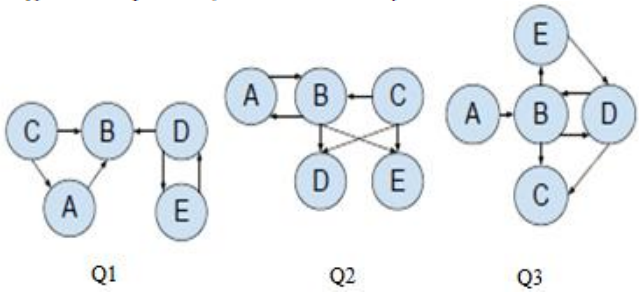

Figure 2: Structures of graph representing query Q1, Q2 and Q3.

The greedy and bushy approach have time complexity $O(|V|^2 .C^2)$ and $O(|V|^2 .C!)$, respectively, where C is the number of cycles in the query.

5. Conclusion

In this paper, we presented optimization technique for distributed graph pattern matching and optimization framework that is based on cycle detection. These frameworks explore greedy and bushy plans and the algorithm to reduce cycles in graph and it brought the query optimization also it eliminates redundant sub query pattern matching and reduces network traffic. With this technique, system is able to perform greedy query optimization in graph model.

References

[1] Shuai Ma, Yang Cao, Jinpeng Huai, Tianyu Wo. Distributed Graph Pattern Matching **(**2012)
[2] S. Chaudhuri. An overview of query optimization in relational systems. PODS '98, pages 34–43(1998)
[3] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In ICDE, pages 913–922, (2008)
[4] J. Cheng, J. X. Yu, and P. S. Yu. Graph pattern matching: A join/semijoin approach. IEEE TKDE., July (2011)
[5] W. Fan. Graph pattern matching revised for social network analysis. In ICDT, pages 8–21 (2012)
[6] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. SIGMOD Rec. (1992)
[7] H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In SIGMOD (2008)
[8] W. Hong and M. Stonebraker. Optimization of parallel query execution plans in xprs. PDIS (1991)
[9] L. Zou, L. Chen, and M. T. O¨ zsu. Distancejoin: Pattern match query in a large graph database. PVLDB, 2(1):886–897, (2009)
[10] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. In WWW, pages 949–958 (2012)
[11] Charu C Aggarwal and Haixun Wang. Managing and mining graph data, volume 40. Springer, (2010)
[12] Horst Bunke and G Allermann. Inexact graph matching for structural pattern recognition. Pattern Recognition Letters, 1(4):245,253. (1983)
[13] G. Cong, W. Fan, and A. Kementsietsidis. Distributed query evaluation with performance guarantees. In SIGMOD, (2007)
[14] M. S. Chen and P. S. Yu. Combining joint and semi-join operations for distributed query processing. *TKDE*, 5(3) (1993)
[15] Arash Fard, M. Usman Nisar, John A. Miller, Lakshmish Ramaswamy**,** Distributed and scalable graph pattern matching: models and algorithms, IJBD (2014)

Author Profile

Rajshri G. Deshmukh received her B.E.(CSE) From P. R. Pote COET, Amravati Affiliated to Sant Gadge Baba Amravati University, Amravati, Maharashtra, India in 2014. Currently pursuing her M.E. in computer science and engineering at P. R. Pote COET,

Amravati Affiliated to Sant Gadge Baba Amravati University, Amravati, and Maharashtra, India.

Praful B. Sambhare Working as Assistant Professor in the Department of Computer Science and Engineering, P. R. Pote COET, Amravati. He completed his M.Tech (CSE) in 2014 from PIES, Bhopal, MP, and India and guided several projects.